19 | 性能优化六大原则：三要三不要，快速有效地进行优化

2020-01-08 庄振运

性能工程高手课 进入课程 >

讲述：庄振运
时长 13:04　大小 10.49M

你好，我是庄振运。

今天我们进入了专栏的新模块：性能优化。在这个模块里，我会先从“性能优化的六大原则”开始讲起，然后再为你讲解实践中普遍采用的十个性能优化策略，并且分别针对CPU、系统、存储以及跨层这几个领域，讲讲具体的优化案例。

我们今天先探讨性能优化的原则。在讲具体原则之前，我想先给你讲一个有趣的往事。

我曾经负责过一个存储服务的性能优化和容量效率。那个服务的容量需求很大，但它的最大性能瓶颈不是 CPU，而是存储的空间。

所以，虽然有很多人给我们各种建议，让我们花时间做 CPU 优化，我都尽量挡了回去。因为我知道 CPU 不是最大问题，所以坚持不懈地通过各种途径优化数据大小，甚至以牺牲 CPU 为代价。最后的结果很好，大幅度地降低了那个服务的容量需求。

在这个性能优化的场景，我们遵循了一个原则，那就是**优先优化最大性能瓶颈**。这其实就是马上要讲到的“三要”原则中的第一个。

性能优化的原则概述

在实际的性能优化中，我们需要考虑的因素很多，也经常需要在多个角度和目标间做一些平衡和取舍。为了帮助你把握这些，我个人总结出了六条原则，我把它们概括为：“**三要，三不要**”。

三个“要”原则是：要优先查最大的性能瓶颈，性能分析要确诊性能问题的根因，性能优化要考虑各种的情况。

三个“不要”的原则是：不要做过度的、反常态的优化，不要过早做不成熟的优化，不要做表面的肤浅优化。

接下来，我们一个个地讲一下这六个优化原则。

三个“要”的原则

你可能已经发现了，这三个“要”的原则之间，其实是步步递进的关系。也就是首先需要查找最大性能瓶颈，然后确诊性能瓶颈产生的原因，最后针锋相对地提出最好的解决方案。

而这个最优解，往往是在考虑各种情况之后提出来，并最终被选中的。

要优先查最大的性能瓶颈

任何一个应用程序或者系统，总会有很多地方可以优化。

可是你知道要从何处下手吗？

其实你可以参考古人的观点，唐代的诗人杜甫说过：“射人先射马，擒贼先擒王”。会打仗的将军，一定会从“最值得打的地方”开始，性能优化工作也是如此。

我们永远都要优先从**最大的性能瓶颈**入手。

一般来讲，如果找到最大的性能瓶颈，并且解决了它，那这个系统的性能会得到最大的提升（参见第 4 讲帕累托法则）。反之，如果不解决最大的性能瓶颈，反而退而求其次，去解决了其他的性能问题，整个系统的性能或许会更高一些，但是提升的程度往往是非常有限的。

比如，一个应用程序的最大性能瓶颈是 CPU 的使用率太高；而其他种类型的资源，比如网络和存储，都很有富余。这种情况下，如果你去优化网络和存储方面，显然是不能大幅度地提升整体性能的。

不过这里有个稍微特殊的情况，就是**内存**。因为内存的分配和回收也会消耗一些 CPU 资源，如果这时去优化内存使用，很多时候，的确会帮助你降低 CPU 使用率，不过降低的幅度一定是很有限的。

归根结底，要降低 CPU 的使用率，**最有效的方法是做性能分析和剖析，找出程序中使用 CPU 最多的地方**，然后对症下药地做优化。

要确诊性能问题的根因

我们前面讲过，程序和系统如果在某个地方有性能瓶颈，肯定是这个地方的资源不够用了，不管是 CPU、内存还是网络（参见第 14 讲）。

所以，当我们确定了最大的性能瓶颈后，就需要对这一性能瓶颈做彻底的性能分析，找出资源不够使用的原因，也就是考察使用资源的地方。

一种资源被使用的地方往往有好几个，我们需要一个一个地去分析考虑。只有彻底分析了各种使用的情况，才能进一步找出最主要的，也是最可能优化的原因，对症下药。

有些资源使用的原因也许是完全合理的。对这些合理的使用，有些或许已经仔细优化过了，很难再做优化。而另外一些则有可能继续优化。对资源的不合理使用，我们就要尽量想办法去掉。

对于需要优化的地方，我们就需要进一步考虑**优化工作的投入产出比例**，就是**既考虑成本，也考虑带来的好处**。因为有些情况下，虽然你可以去优化，但获得的收益并不大，所以不值得去做。

另外要提醒你的是，确诊性能问题的原因有时候非常困难，需要做多方面的性能测试、假设分析并验证。

比如 CPU 的使用，有操作系统的原因，有应用程序的原因，但也有些 CPU 的问题，是在非常边缘的场景下才发生的。为了暴露问题，我们经常需要创造特殊的场景来重现遇到的性能问题。

要考虑各种情况下的性能

性能问题确诊原因后，我们就可以进入下一步，找解决方案了。一般说来，找一个解决方案并不难，甚至找好几个方案也不难；但是找出一个好的、最优的解决方案是真心不容易。

为什么这么说呢？

因为实际生产环境很复杂，而且会出现各种各样的特殊场景。

针对某个具体场景提出的一个解决方案，多半并不能适应所有的场景。

所以，对提出的各种方案进行评估时，我们必须考虑各种情况下这个方案可能的表现。如果一个方案在某些情况下会导致其他严重的问题，这个方案或许就不是一个好的方案。

但同时你也需要意识到，任何解决方案都有长短，有 Tradeoff。如果苛求一个能在所有场景下都最优的解决方案，往往是缘木求鱼，是不现实的。

比如一种优化方案，可以让平均响应时间最小，但高百分位比较高。另外一种优化方案正好相反。那我们就需要考虑对自己来说哪种指标更重要。也就是说，我们经常需要在**不同性能指标间权衡**，以找到一个最优解能**达到总体和整体最优**。

这就需要我们**有一个整体的意识和判断**。

或许一个方案并不能面面俱到，在有些场景下性能不好。但是不同场景的出现概率不同，对其它模块造成的影响也不一样，并且最终客户的体验也不尽相同。这些因素都要考虑到取舍的决策过程中。

综合以上因素，在实际的优化过程中，我们经常会**反复权衡利弊和取舍来做最终决定**。

三个“不要”的原则

讲完了三个“要”的原则，我们接着来看三个“不要”的原则。

不要过度地反常态优化

性能优化的目标，是追求**最合适的性价比**或**最高的投入产出比**，在满足要求的情况下，尽量不要做过度的优化。过度的优化会增加系统复杂度和维护成本，使得开发和测试周期变长。虽然性能上带来了一定程度的提升，但是和导致的缺点来比，孰轻孰重尚不可知，需要仔细斟酌，衡量得失。

我的建议是，**根据产品的性能要求来决策**。

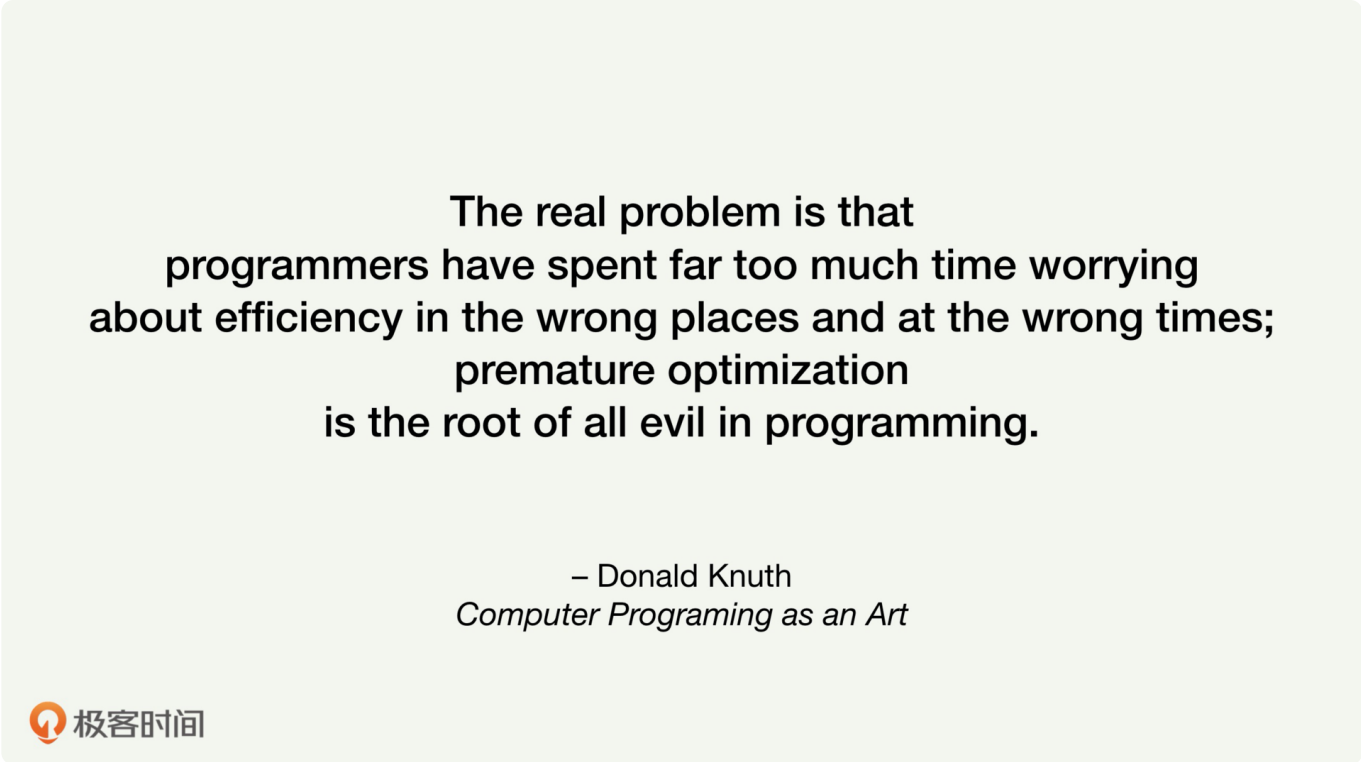
在设计产品时，我们对产品的性能会有一定的要求，比如吞吐量，或者客户响应时间要达到多少多少。如果达不到这个既定指标，就需要去优化。反之，如果能满足这些指标，那么就不必要花费太多时间精力去优化。

比如，我们要设计一个内部查询系统，预计最多只有一百个人同时在线使用的话，就完全不用按照百万在线用户的目标去过度优化。

更重要的是，多数的优化方法是并不是完美无缺的，是有缺点的，尤其是可能会对系统设计的简化性，对代码的可读性和可维护性有副作用。如果系统简化性和代码可读性更加重要，当然就更不能过度优化。

不要过早的不成熟优化

要体会这一原则，我们先引用著名计算机科学家高德纳（Donald Knuth）的一段话：“现实中的最大问题是，程序员往往花太多时间，来在错误的地方和错误的时间来试图提高效率和性能。过早的优化，是编程中所有邪恶和悲剧（或至少是大多数邪恶和悲剧）的根源。”



你只要稍微思考一下高德纳的话，就会发现，这句话在很多场景下都是很有道理的。

比如，在敏捷开发过程中，尤其是在面对一个全新的产品时，在业界没有先例和经验可遵循的情况下，最看重的特点是快速的迭代与试错，“尽快推出产品”是最重要的。这时，过早的优化很可能优化错地方，也就是优化的地方并非真正的性能瓶颈，因此让“优化工作”成为了无用功。而且，越早的优化就越容易造成负面影响，比如影响代码的可读性和维护性。

我个人认为，**如果一个产品已经在业界很成熟**，大家非常清楚它的生产环境特点和性能瓶颈，那么**优化的重要性可以适当提高**。否则的话，在没有实际数据指标的基础上，为了一点点的性能提升而进行盲目优化，的确是得不偿失的。

不要表面的肤浅优化

性能优化很忌讳表面和肤浅的优化，也就是那种“头痛医头，脚痛医脚”的所谓“优化”。如果对一个程序和服务没有全局的把握，没有理解底层运行机制，任何优化方案都很难达到最好的优化效果。

比如，如果你发现一个应用程序的 CPU 使用率并不高，但是吞吐率上不去，表面的优化方式可能是增大线程池来提升 CPU 使用率。这样的简单“优化”或许当时能马上看到效果，比如吞吐率也上去了，但是如果你仔细想想，就会发现如此的表面优化非常有问题。

这样的情况下，线程池开多大最合适？需不需要根据底层硬件和上层请求的变化而对线程池的大小调优呢？如果需要，那么手工调整线程池大小就是一个典型的“头痛医头”的优化。

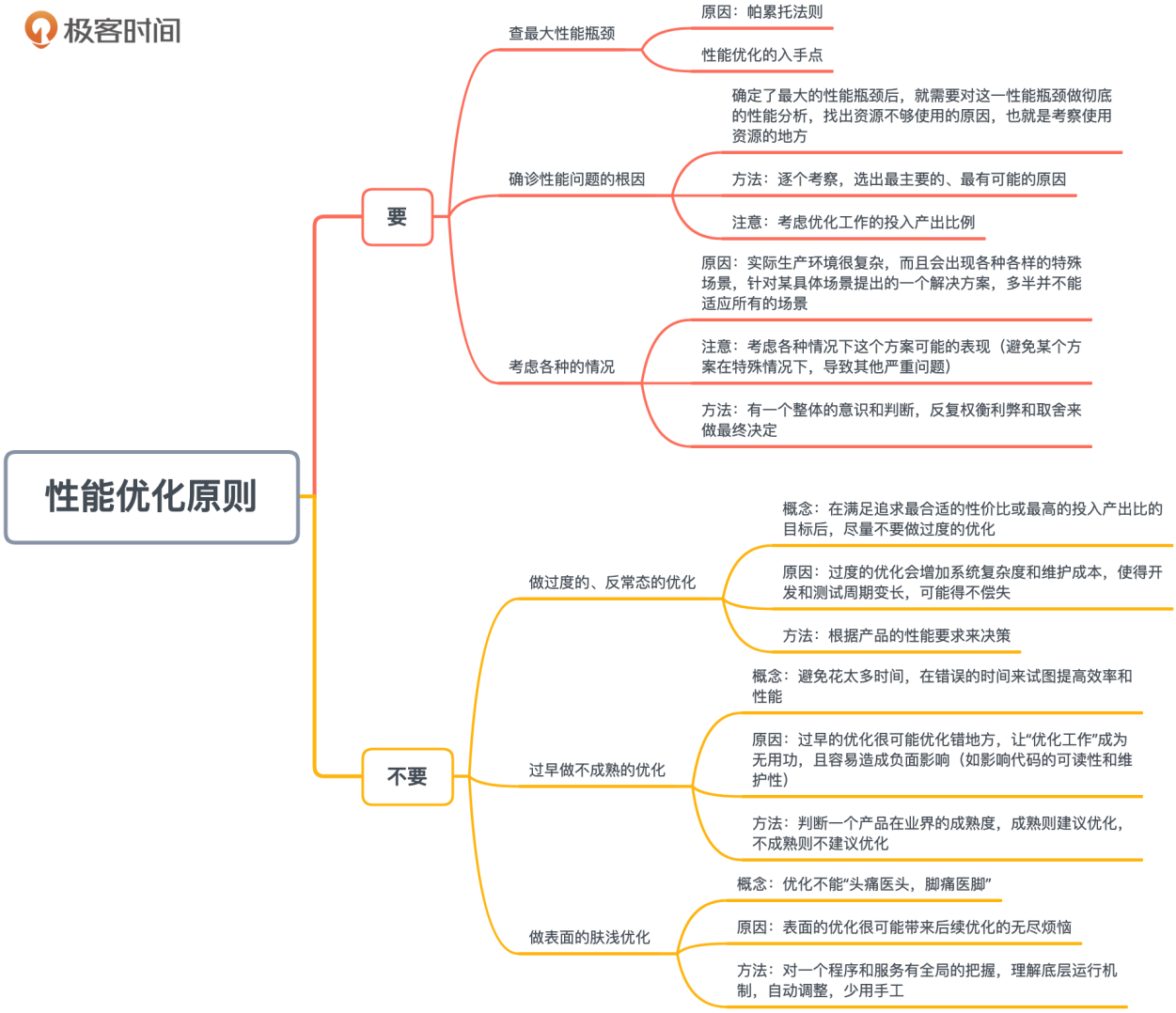
为什么呢?

因为部署环境不会一成不变，比如以后 CPU 升级了，核数变多了，你怎么办？再次手工去调整吗？这样做很快会让人疲于奔命，难以应付，并且很容易出错。

对这样的场景，正确的优化方式，是彻底了解线程的特性，以优化线程为主。至于线程池的大小，最好能够自动调整。千万别动不动就手工调优。如果这样手工调整的参数多了，就会做出一个有很多可调参数的复杂系统，很难用，也很难调优，很不可取。就比如我们都熟悉的 JVM 调优，有上千个可调参数，非常被人诟病。

总结

唐朝的名相魏征说过：“求木之长者，必固其根本”。意思是说，如果要一棵树长得高，必须让它的根牢固。否则的话，正如魏征自己所说：“根不固而求木之长”，一定是“知其不可”。根基不牢固，就不可能长成参天大树。



同理，对现代互联网的服务和系统来说，性能问题是根本的问题。如果不知道系统的性能瓶颈，查不出性能根因，不知道如何解决，无法做合理的优化，这个服务和系统一定不会高效。

这一讲我们总结了六个性能优化的原则，这些原则的终极目的，就是找出性能的最大瓶颈，查出根因，并作做相应的最优优化，从而让我们的系统这棵树长高、长大。

思考题

今天介绍的六大原则里面，你认为哪个原则最重要？为什么？

你过去的工作中有没有因为没有遵循这几个原则而吃后悔药的时候？比如你在选择一个数据结构的具体实现时（比如 Set），有没有考虑各种场景下的性能？

欢迎你在留言区分享自己的思考，与我和其他同学一起讨论，也欢迎你把文章分享给自己的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　18 | 网络篇：如何步步拆解处理复杂的网络性能问题?

下一篇　20 | 性能优化十大策略：如何系统地有层次地优化性能问题?

精选留言 (1)

写留言

Wang
2020-01-09

庄老师，近期在验证tomcat 8 在NIO和APR两种模式下的性能差异。理论上APR在高并发的情况下性能更好一些，但是我验证的结果却是NIO的性能更好（同样的并发线程数，NIO模式的吞吐量更高，相同的吞吐量，系统的资源利用率相差并不大）。代码中只是设置了等待20或200ms然后打印输出字符，无任何逻辑。有些迷茫求庄老师帮忙分析一下，多谢🙏

展开

作者回复: Tomcat我不熟. :(抱歉帮不上，你试试google一下？并且比较一下各个性能指标看看有没有hint?